SIAM Conference on Computational Science and Engineering

Short Course on the ACTS Collection:

Robust and High Performance Libraries for Computational Sciences

ScaLAPACK

(introduction)

Osni Marques

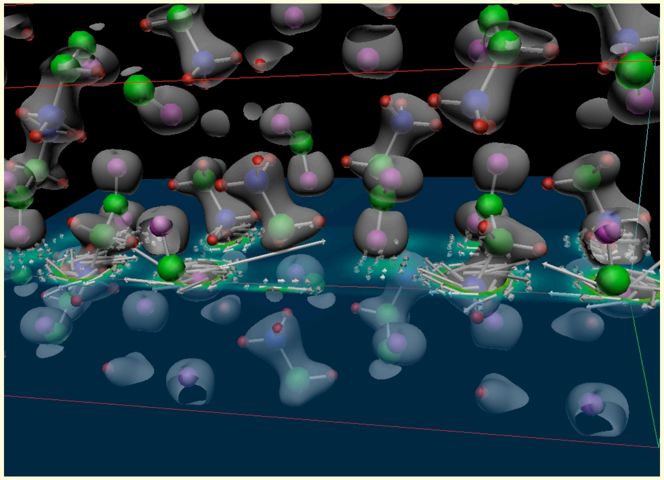
Lawrence Berkeley National Laboratory (LBNL)

oamarques@lbl.gov

Outline

- Motivation (some applications)
- ScaLAPACK: *software structure*
  - Basic Linear Algebra Subprograms (BLAS)
  - Linear Algebra PACKage (LAPACK)
  - Basic Linear Algebra Communication Subprograms (BLACS)
  - Parallel BLAS (PBLAS)
- ScaLAPACK: *details*
  - Data layout
  - Array descriptors
  - Error handling
  - Performance
- Examples

Applications

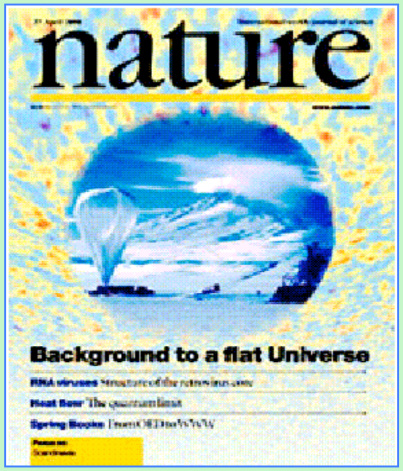


Induced current (white arrows) and charge density (colored plane and gray surface) in crystallized glycine due to an external field (Louie, Yoon, Pfrommer and Canning), eigenvalue problems solved with ScaLAPACK.

Advanced Computational Research in Fusion (SciDAC Project, PI Mitch Pindzola). Point of contact: Dario Mitnik (Dept. of Physics, Rollins College). Mitnik attended the workshop on the ACTS Collection in September 2000. Since then he has been actively using some of the ACTS tools, in particular ScaLAPACK, for which he has provided insightful feedback. Dario is currently working on the development, testing and support of new scientific simulation codes related to the study of atomic dynamics using time-dependent close coupling lattice and time-independent methods. He reports that this work could not be carried out in sequential machines and that ScaLAPACK is fundamental for the parallelization of these codes.

Performance of four science-of-scale applications that use ScaLAPACK functionalities on an IBM SP

Project	Number of processors	Performance (% of peak)
Electromagnetic Wave-Plasma Interactions	1,936	68%
Cosmic Microwave Background Data Analysis	4,096	50%
Terascale Simulations of Supernovae	2,048	43%
Quantum Chromodynamics at High Temperatures	1,024	13%



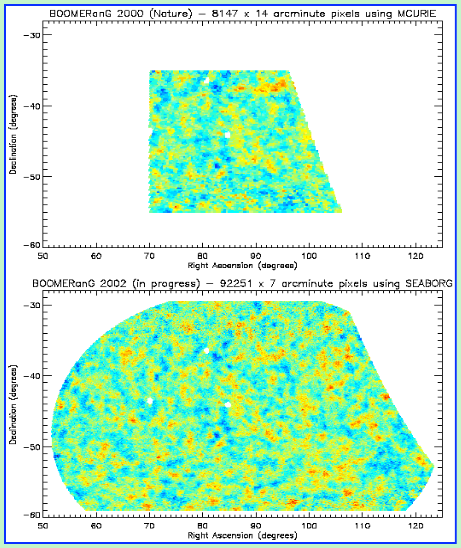
nature

Background to a flat Universe

RNA viruses

Heat flow The quantum limit

Spring Books


BOOMERanG 2000 (Nature) – 8147 x 14 arcminute pixels using MCURIE
Declination (degrees)
−30
−40
−50
−60
50
60
70
80
90
100
110
120
Right Ascension (degrees)
BOOMERanG 2002 (in progress) – 92251 x 7 arcminute pixels using SEABORG
Declination (degrees)
−30
−40
−50
−60
50
60
70
80
90
100
110
120
Right Ascension (degrees)

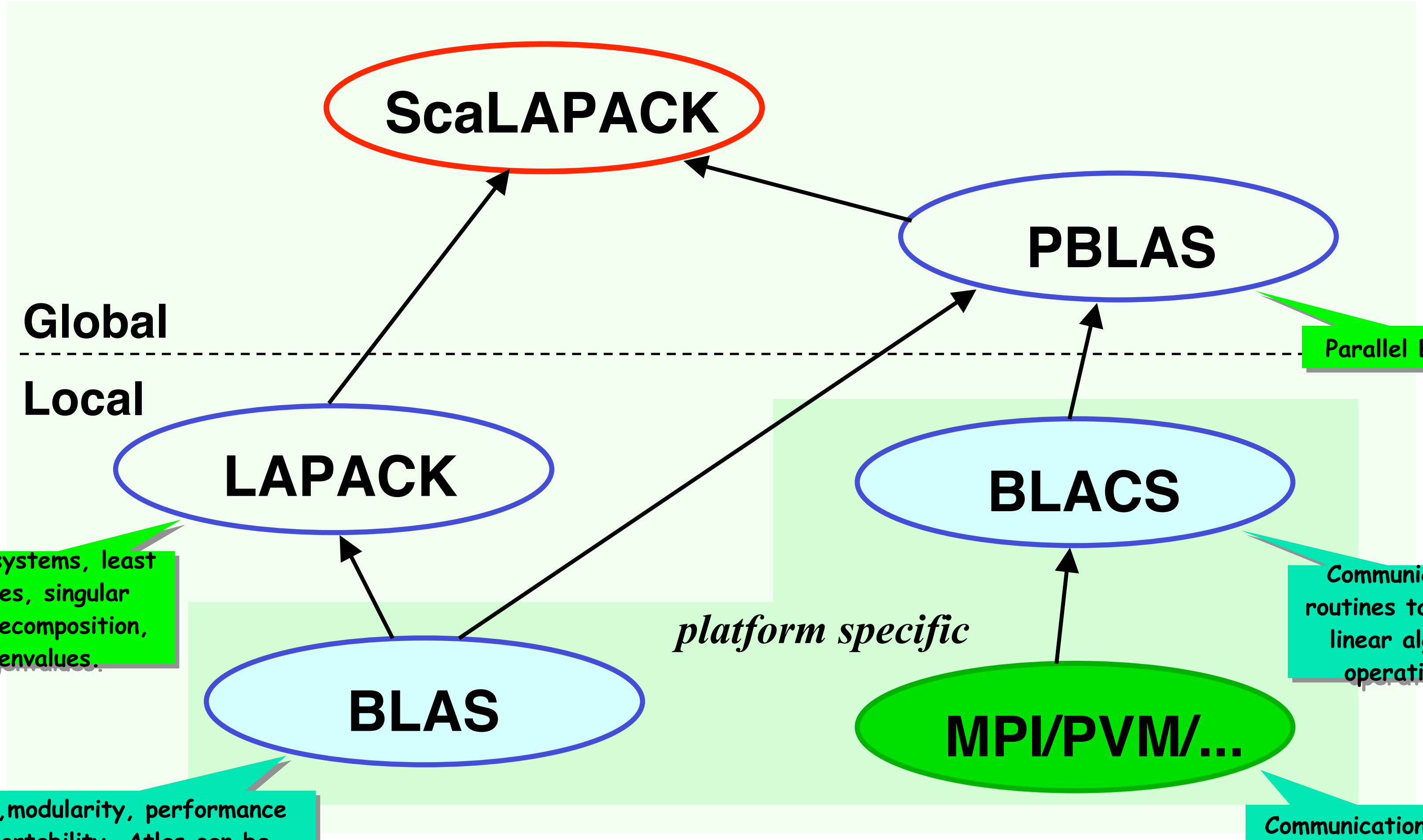
The international BOOMERanG collaboration announced results of the most detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat (Apr. 27, 2000). Likelihood methods implemented in the MADCAP software package, using routines from ScaLAPACK, were used to examine the large dataset generated by BOOMERanG.

Cosmic Microwave Background (CMB) Analysis

- The CMB is the faint echo of the Big Bang.
- The statistics of the tiny variations in the CMB allows the determination of the fundamental parameters of cosmology to the percent level or better.
- MADCAP (Microwave Anisotropy Dataset Computational Analysis Package)
  - Makes maps from observations of the CMB and then calculates their angular power spectra. (See *http://crd.lbl.gov/~borrill*).
  - Calculations are dominated by the solution of linear systems of the form $M = A^{-1}B$ for dense nxn matrices A and B scaling as O(n^3) in flops. MADCAP uses ScaLAPACK for those calculations.
- On the NERSC Cray T3E (original code):
  - Cholesky factorization and triangular solve.
  - Typically reached 70-80% peak performance.
  - Solution of systems with $n \sim 10^4$ using tens of processors.
  - The results demonstrated that the Universe is spatially flat (overall) comprising 70% dark energy, 25% dark matter, and only 5% ordinary matter.
- On the NERSC IBM SP:
  - Porting was trivial but tests showed only 20-30% peak performance.
  - Code rewritten to use Cholesky factorization, triangular matrix inversion and triangular matrix multiplication ➡ one-day work because of the completeness and coherence of ScaLAPACK.
  - Performance increased to 50-60% peak.
  - Solution of previously intractable systems with $n \sim 10^5$ using hundreds of processors.

ScaLAPACK: *software structure*

http://acts.nersc.gov/scalapack



Parallel BLAS.

Linear systems, least squares, singular value decomposition, eigenvalues.

Communication routines targeting linear algebra operations.

Clarity,modularity, performance and portability. Atlas can be used here for automatic tuning.

Communication layer (message passing).

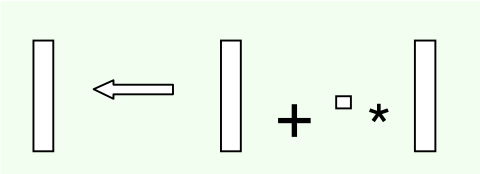
BLAS

(*Basic Linear Algebra Subroutines*)

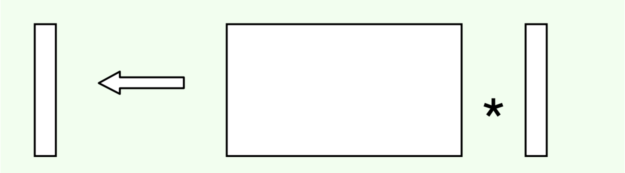
- Clarity: code is shorter and easier to read.
- Modularity: gives programmer larger building blocks.
- Performance: manufacturers (usually) provide tuned machine-specific BLAS.
- Portability: machine dependencies are confined to the BLAS.
- Key to high performance: effective use of memory hierarchy (true on all architectures).

BLAS: *3 levels*

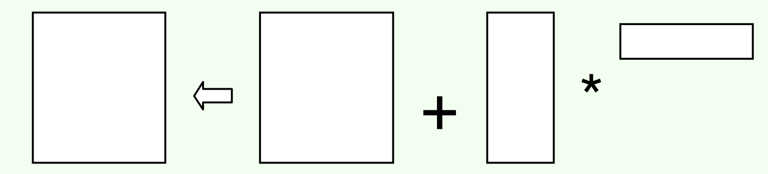
- Level 1 BLAS:
  vector-vector operations.



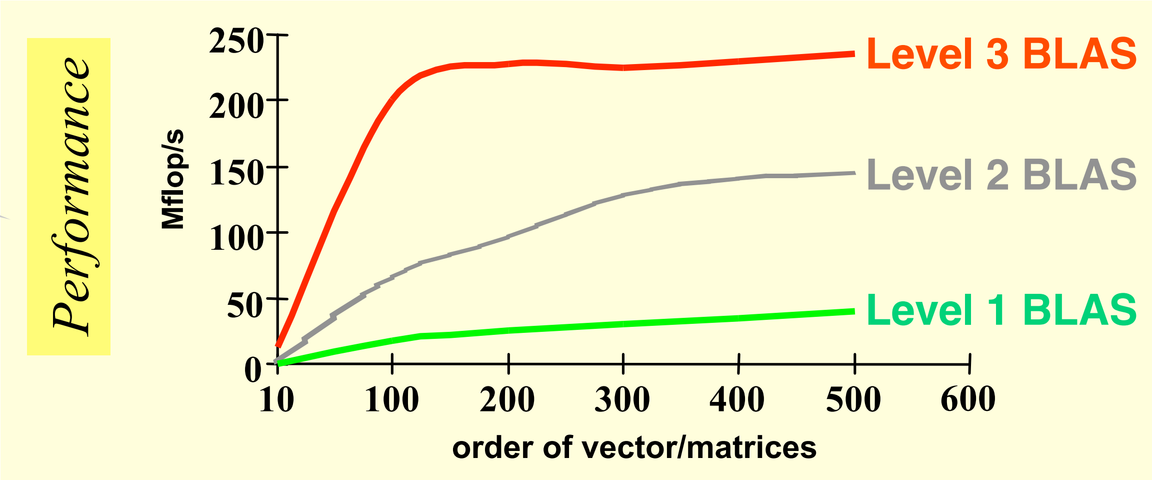
- Level 2 BLAS:
  matrix-vector operations.



- Level 3 BLAS:
  matrix-matrix operations.



Development of blocked algorithms (BLAS 3) is important for performance!



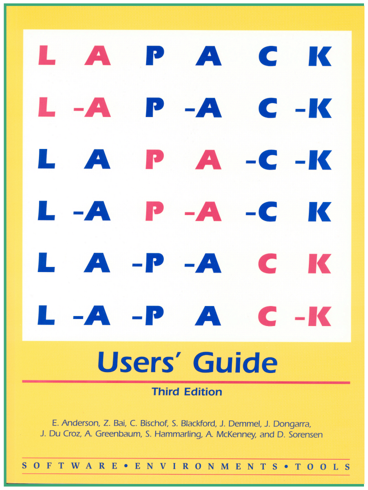
LAPACK

(*http://www.netlib.org/lapack*)

- Linear Algebra library written in Fortran 77 (Fortran 90, C and C++ versions also available).
- Combine algorithms from LINPACK and EISPACK into a single package.
- Efficient on a wide range of computers (RISC, Vector, SMPs).
- User interface similar to LINPACK (Single, Double, Complex, Double Complex).
- Built atop level 1, 2, and 3 BLAS for high performance, clarity, modularity and portability.

LAPACK: *features*

- Basic problems:
  - Linear systems: $Ax = b$
  - Least squares: $\min \|Ax - b\|_2$
  - Singular value decomposition: $A = U\Sigma V^T$
  - Eigenvalues and eigenvectors: $Az = \lambda z, \quad Az = \lambda Bz$
- LAPACK does not provide routines for structured problems or general sparse matrices (i.e. sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).
- LAPACK Users' Guide, Third Edition (1999)



L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K

Users' Guide

Third Edition

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra,
J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen

SOFTWARE • ENVIRONMENTS • TOOLS

BLACS

(*Basic Linear Algebra Communication Subroutines*)

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations. This improves:
  - program readability
  - self-documenting quality of the code.
- Promote efficiency by identifying frequently occurring operations of linear algebra which can be optimized on various computers.

BLACS: *basics*

- Processes are embedded in a two-dimensional grid.

Example: a 3x4 grid

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

- An operation which involves more than one sender and one receiver is called a *scoped operation*.

Scope	Meaning
Row	All processes in a process row participate.
Column	All processes in a process column participate.
All	All processes in the process grid participate.

BLACS: *communication routines*

***Send/Receive*:**

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix TR: Trapezoidal matrix

***Broadcast*:**

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)

_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

SCOPE	TOP
‘Row’ ‘Column’ ‘All’	‘ ‘ (default) ‘Increasing Ring’ ‘1-tree’ ...

BLACS: *global combine operations*

- Perform element-wise SUM, |MAX|, |MIN|, operations on triangular matrices:

```
_GSUM2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
_GAMX2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
_GAMN2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
```

- RDEST = –1 indicates that the result of the operation should be left on all processes selected by SCOPE.
- For |MAX|, |MIN|, when RCFLAG = –1, RA and CA are not referenced; otherwise RA and CA are set on output with the coordinates of the process owning the corresponding maximum (or minimum) element in absolute value of A.

BLACS: *example*

```
*   Get system information
    CALL BLACS_PINFO( IAM, NPROCS )
*   Get default system context
    CALL BLACS_GET( 0, 0, ICTXT )
    M
*   Define 1 x (NPROCS/2+1) process grid
    NPROW = 1
    NPCOL = NPROCS / 2 + 1
    CALL BLACS_GRIDINIT( ICTXT, 'Row', NPROW, NPCOL )
    CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*   If I'm not in the grid, go to end of program
    IF( MYROW.NE.-1 ) THEN
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
        CALL DGESD2D( ICTXT, 5, 1, X, 5, 1, 0 )
      ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
        CALL DGERV2D( ICTXT, 5, 1, Y, 5, 0, 0 )
      END IF
       M
      CALL BLACS_GRIDEXIT( ICTXT )
    END IF
    CALL BLACS_EXIT( 0 )
    END
```

(out) uniquely identifies each process

(out) number of processes available

(in) integer handle indicating the context

(in) use (default) system context

(out) BLACS context

(output) process row and column coordinate

send X to process (1,0)

receive X from process (0,0)

leave context

exit from the BLACS

- **The BLACS context is the BLACS mechanism for partitioning communication space.**
- **A message in a context cannot be sent or received in another context.**
- **The context allows the user to**
  - **create arbitrary groups of processes**
  - **create multiple overlapping and/or disjoint grids**
  - **isolate each process grid so that grids do not interfere with each other**
- **BLACS context ⇔ MPI communicator**

See *http://www.netlib.org/blacs* for more information.

PBLAS

(*Parallel Basic Linear Algebra Subroutines*)

- Similar to the BLAS in portability, functionality and naming.
- Built atop the BLAS and BLACS
- Provide global view of matrix

```
CALL DGEXXX( M, N, A( IA, JA ), LDA, ... )
```

BLAS

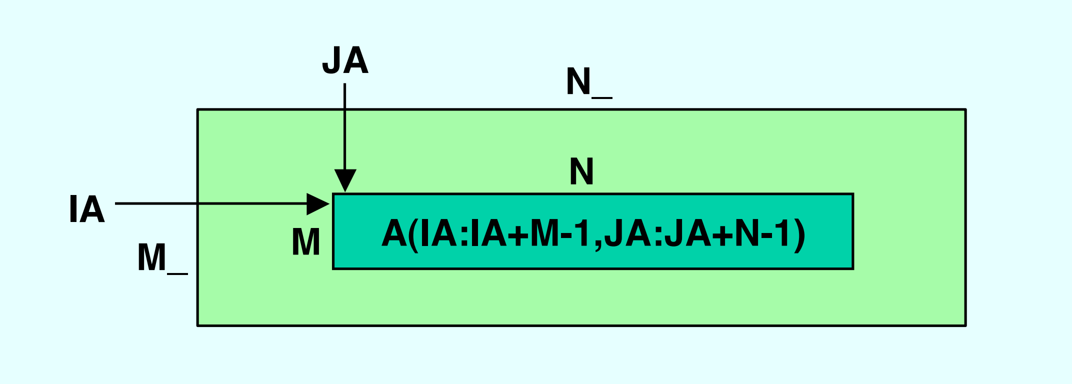
```
CALL PDGEXXX( M, N, A, IA, JA, DESCA, ... )
```

PBLAS

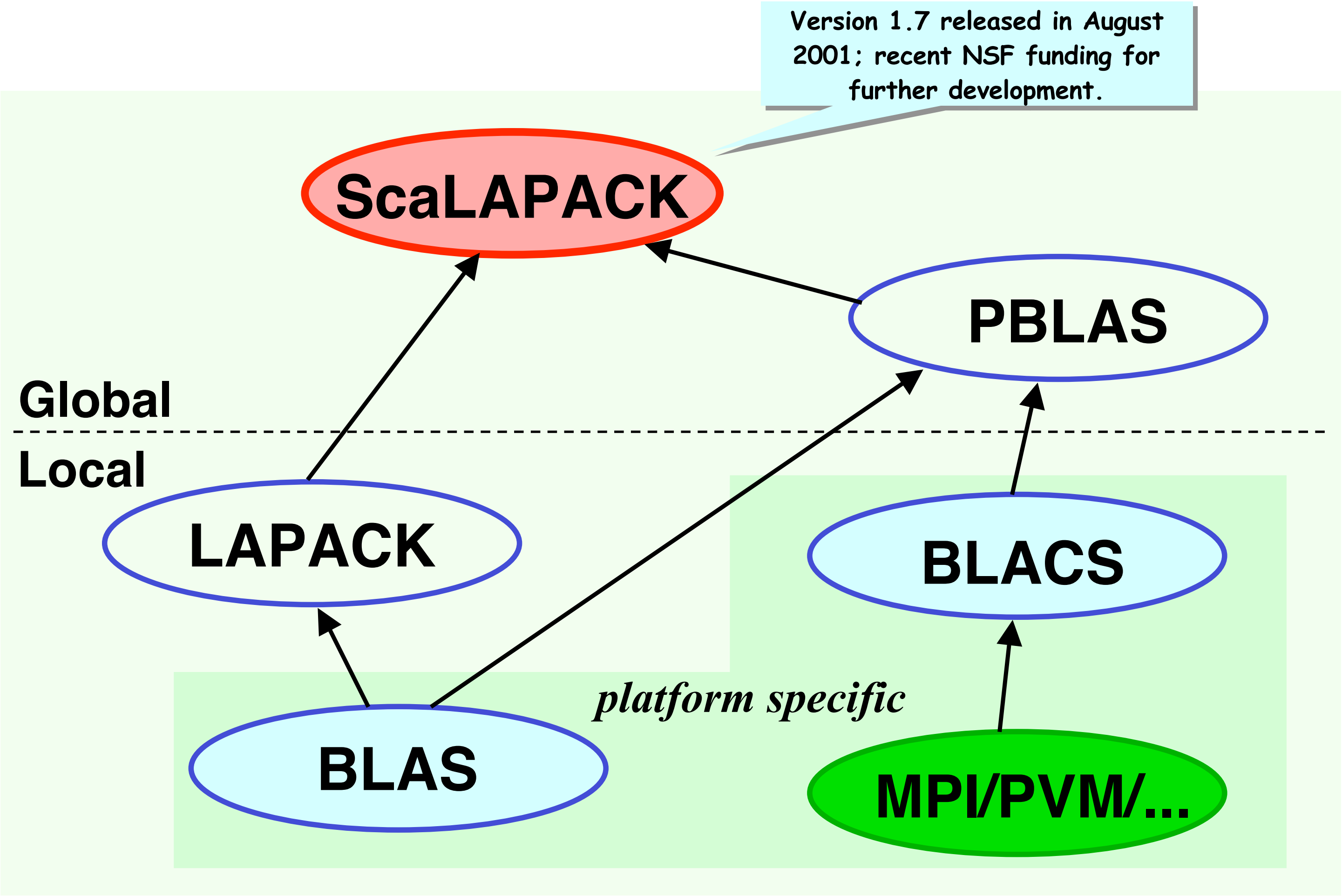
Array descriptor (see next slides)

PBLAS: *levels and view of the operands*

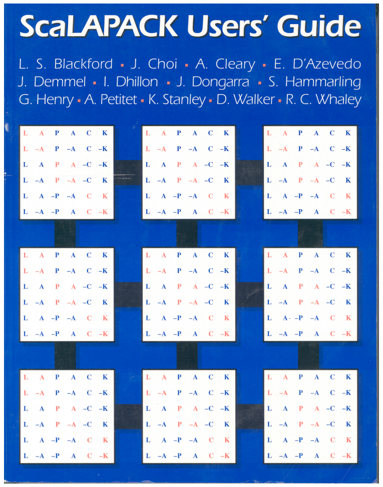
- Levels:
  - Level 1: vector-vector operations.
  - Level 2: matrix-vector operations.
  - Level 3: matrix-matrix operations.
- Global view of the matrix operands, allowing global addressing of distributed matrices (hiding complex local indexing)



ScaLAPACK: *structure of the software*



Version 1.7 released in August 2001; recent NSF funding for further development.



ScaLAPACK Users' Guide

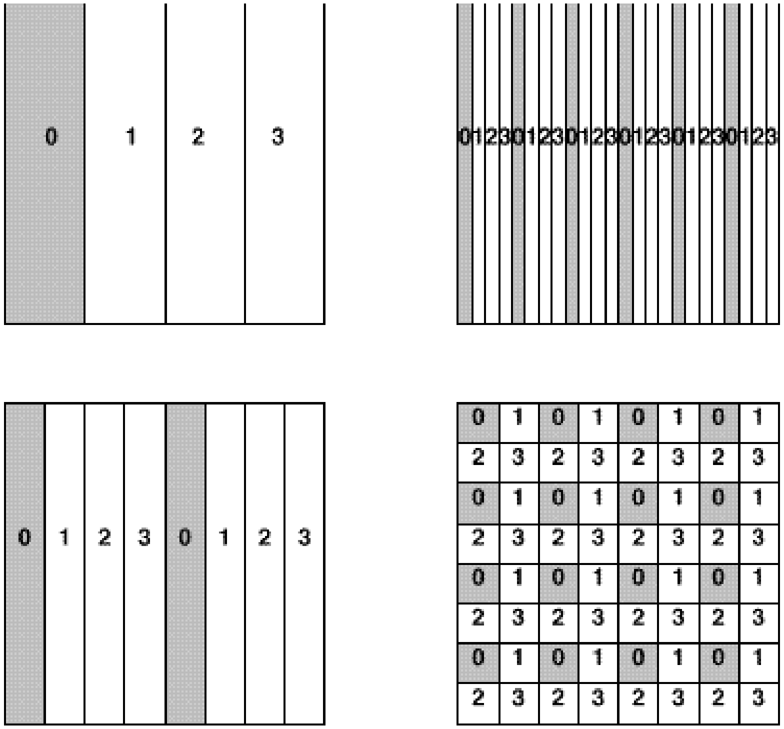
L. S. Blackford • J. Choi • A. Cleary • E. D'Azevedo
J. Demmel • I. Dhillon • J. Dongarra • S. Hammarling
G. Henry • A. Petitet • K. Stanley • D. Walker • R. C. Whaley

ScaLAPACK: *goals*

- Efficiency
  - Optimized computation and communication engines
  - Block-partitioned algorithms (Level 3 BLAS) for good node performance
- Reliability
  - Whenever possible, use LAPACK algorithms and error bounds
- Scalability
  - As the problem size and number of processors grow
  - Replace LAPACK algorithm that did not scale (new ones into LAPACK)
- Portability
  - Isolate machine dependencies to BLAS and the BLACS
- Flexibility
  - Modularity: build rich set of linear algebra tools (BLAS, BLACS, PBLAS)
- Ease-of-Use
  - Calling interface similar to LAPACK

ScaLAPACK: *data layouts*

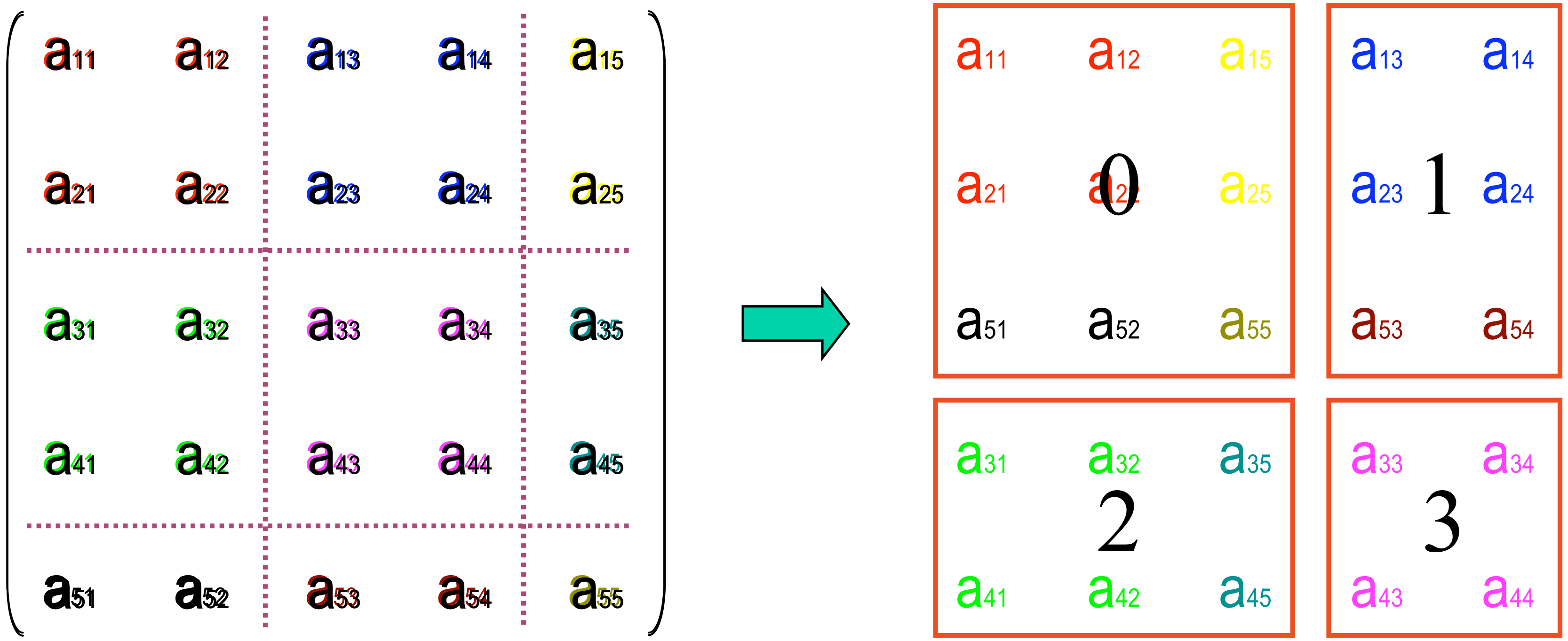
- 1D block and cyclic column distributions



0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

ScaLAPACK: *2D Block-Cyclic Distribution*



5x5 matrix partitioned in 2x2 blocks

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

2x2 process grid point of view

0

$$\begin{matrix} a_{11} & a_{12} & a_{15} \\ a_{21} & a_{22} & a_{25} \\ a_{51} & a_{52} & a_{55} \end{matrix}$$

1

$$\begin{matrix} a_{13} & a_{14} \\ a_{23} & a_{24} \\ a_{53} & a_{54} \end{matrix}$$

2

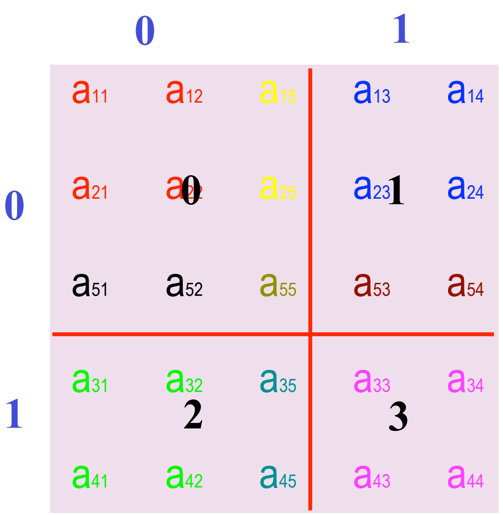
$$\begin{matrix} a_{31} & a_{32} & a_{35} \\ a_{41} & a_{42} & a_{45} \end{matrix}$$

3

$$\begin{matrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{matrix}$$

2D Block-Cyclic Distribution

$$\begin{bmatrix} 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\ -2.1 & 2.2 & 2.3 & 2.4 & 2.5 \\ -3.1 & -3.2 & 3.3 & 3.4 & 3.5 \\ -4.1 & -4.2 & -4.3 & 4.4 & 4.5 \\ -5.1 & -5.2 & -5.3 & -5.4 & 5.5 \end{bmatrix}$$



	0			1	
0	a_{11}	a_{12}	a_{15}	a_{13}	a_{14}
	a_{21}	a_{22}	a_{25}	a_{23}	a_{24}
	a_{51}	a_{52}	a_{55}	a_{53}	a_{54}
1	a_{31}	a_{32}	a_{35}	a_{33}	a_{34}
	a_{41}	a_{42}	a_{45}	a_{43}	a_{44}

```
  M

CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

IF      ( MYROW.EQ.0  .AND.  MYCOL.EQ.0 ) THEN
          A(1) = 1.1; A(2) = -2.1; A(3) = -5.1;
          A(1+LDA) = 1.2; A(2+LDA) = 2.2; A(3+LDA) = -5.2;
          A(1+2*LDA) = 1.5; A(2+3*LDA) = 2.5; A(3+4*LDA) = -5.5;
ELSE IF ( MYROW.EQ.0  .AND.  MYCOL.EQ.1 ) THEN
          A(1) = 1.3; A(2) = 2.3; A(3) = -5.3;
          A(1+LDA) = 1.4; A(2+LDA) = 2.4; A(3+LDA) = -5.4;
ELSE IF ( MYROW.EQ.1  .AND.  MYCOL.EQ.0 ) THEN
          A(1) = -3.1; A(2) = -4.1;
          A(1+LDA) = -3.2; A(2+LDA) = -4.2;
          A(1+2*LDA) = 3.5; A(2+3*LDA) = 4.5;
ELSE IF ( MYROW.EQ.1  .AND.  MYCOL.EQ.1 ) THEN
          A(1) = 3.3; A(2) = -4.3;
          A(1+LDA) = 3.4; A(2+LDA) = 4.4;
END IF
   M
CALL PDGESVD( JOBU, JOBVT, M, N, A, IA, JA, DESCA, S, U, IU,
              JU, DESCU, VT, IVT, JVT, DESCVT, WORK, LWORK,
              INFO )
   M
```

LDA is the leading dimension of the local array (see next slides)

Array descriptor for A (see next slides)

2D Block-Cyclic Distribution

- Ensures good load balance → performance and scalability (analysis of many algorithms to justify this layout).
- Encompasses a large number of data distribution schemes (but not all).
- Needs redistribution routines to go from one distribution to the other.
- See *http://acts.nersc.gov/scalapack/hands-on/datadist.html*

ScaLAPACK: *array descriptors*

- Each global data object is assigned an *array descriptor*.
- The *array descriptor*:
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location (uses concept of BLACS context).
  - Is differentiated by the DTYPE_ (first entry) in the descriptor.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
- User must distribute all global arrays prior to the invocation of a ScaLAPACK routine, for example:
  - Each process generates its own submatrix.
  - One processor reads the matrix from a file and send pieces to other processors (may require message-passing for this).

Array Descriptor for Dense Matrices

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=1 for dense matrices.
2	CTXT_A	(global)	BLACS context handle.
3	M_A	(global)	Number of rows in global array A.
4	N_A	(global)	Number of columns in global array A.
5	MB_A	(global)	Blocking factor used to distribute the rows of array A.
6	NB_A	(global)	Blocking factor used to distribute the columns of array A.
7	RSRC_A	(global)	Process row over which the first row of the array A is distributed.
8	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
9	LLD_A	(local)	Leading dimension of the local array.

Array Descriptor for Narrow Band Matrices

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=501 for 1 x Pc process grid for band and tridi agonal matrices block-column distributed.
2	CTXT_A	(global)	BLACS context handle.
3	N_A	(global)	Number of columns in global array A.
4	NB_A	(global)	Blocking factor used to distribute the columns of array A.
5	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
6	LLD_A	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored.
7	–	–	Unused, reserved.

Array Descriptor for Right Hand Sides for Narrow Band Linear Solvers

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_B	(global)	Descriptor type DTYPE_B=502 for Pr x 1 process grid for block-row distributed matrices .
2	CTXT_B	(global)	BLACS context handle.
3	M_B	(global)	Number of rows in global array B
4	MB_B	(global)	Blocking factor used to distribute the rows of array B.
5	RSRC_B	(global)	Process row over which the first row of the array B is distributed.
6	LLD_B	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored.
7	–	–	Unused, reserved.

ScaLAPACK: *Functionality*

$Ax = b$	Simple Driver	Expert Driver	Factor	Solve	Inversion	Conditioning Estimator	Iterative Refinement
Triangular				x	x	x	x
SPD	x	x	x	x	x	x	x
SPD Banded	x		x	x			
SPD Tridiagonal	x		x	x			
General	x	x	x	x	x	x	x
General Banded	x		x	x			
General Tridiagonal	x		x	x			
Least Squares	x		x	x			
GQR			x				
GRQ			x				
$Ax = \lambda x$ or $Ax = \lambda Bx$	**Simple Driver**	**Expert Driver**	**Reduction**	**Solution**			
Symmetric	x	x	x	x			
General	x	x	x	x			
Generalized BSPD	x		x	x			
SVD			x	x			

ScaLAPACK: *error handling*

- Driver and computational routines perform *global* and *local* input error-checking.
    - Global checking → synchronization
    - Local checking → validity
- No input error-checking is performed on the auxiliary routines.
- If an error is detected in a PBLAS or BLACS routine program execution stops.

ScaLAPACK: *debugging hints*

- Look at ScaLAPACK example programs.
- Always check the value of INFO on exit from a ScaLAPACK routine.
- Query for size of workspace, LWORK = –1.
- Link to the Debug Level 1 BLACS (specified by BLACSDBGLVL=1 in Bmake.inc).
- Consult errata files on *netlib*:

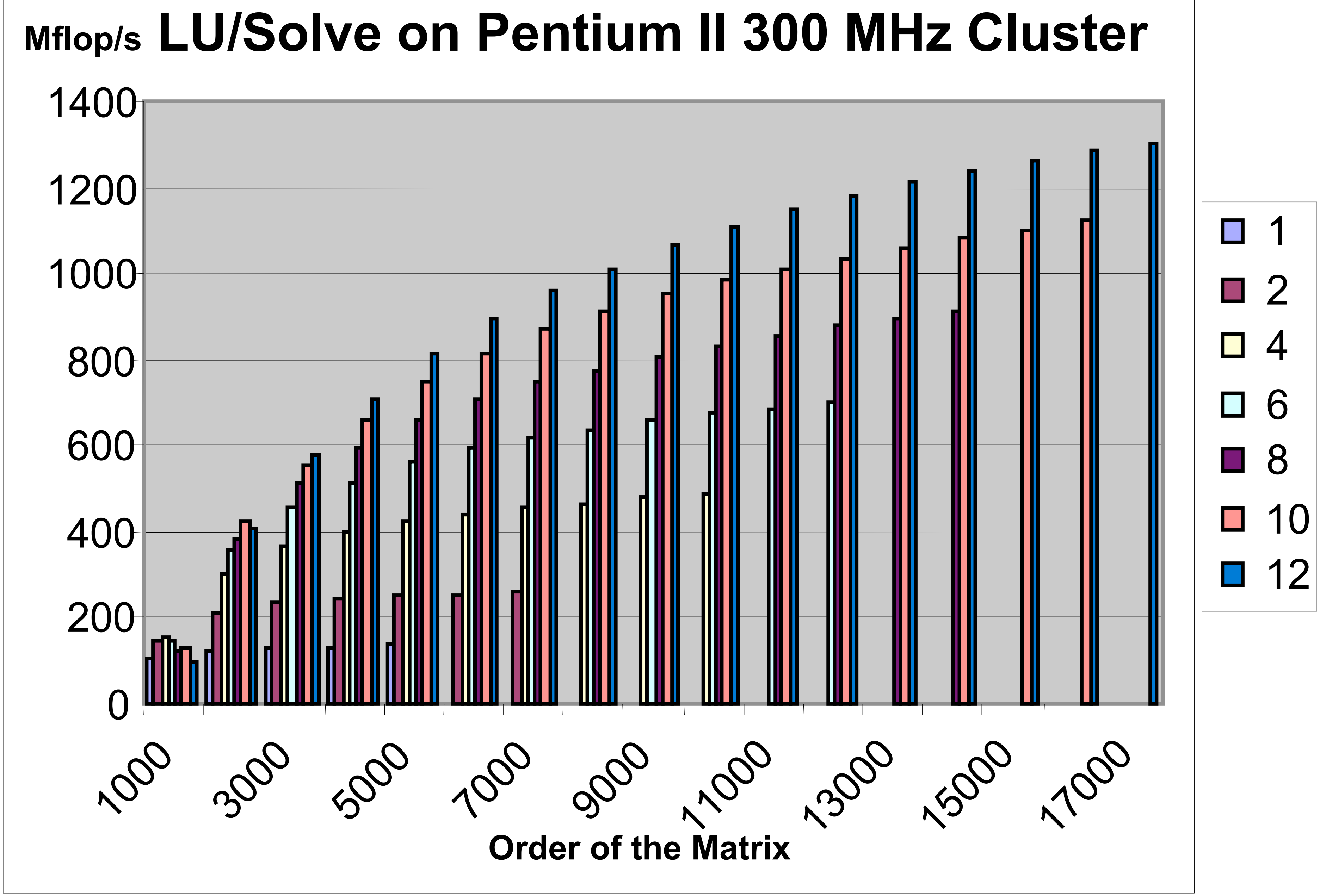
  http://www.netlib.org/scalapack/errata.scalapack

  http://www.netlib.org/blacs/errata.blacs

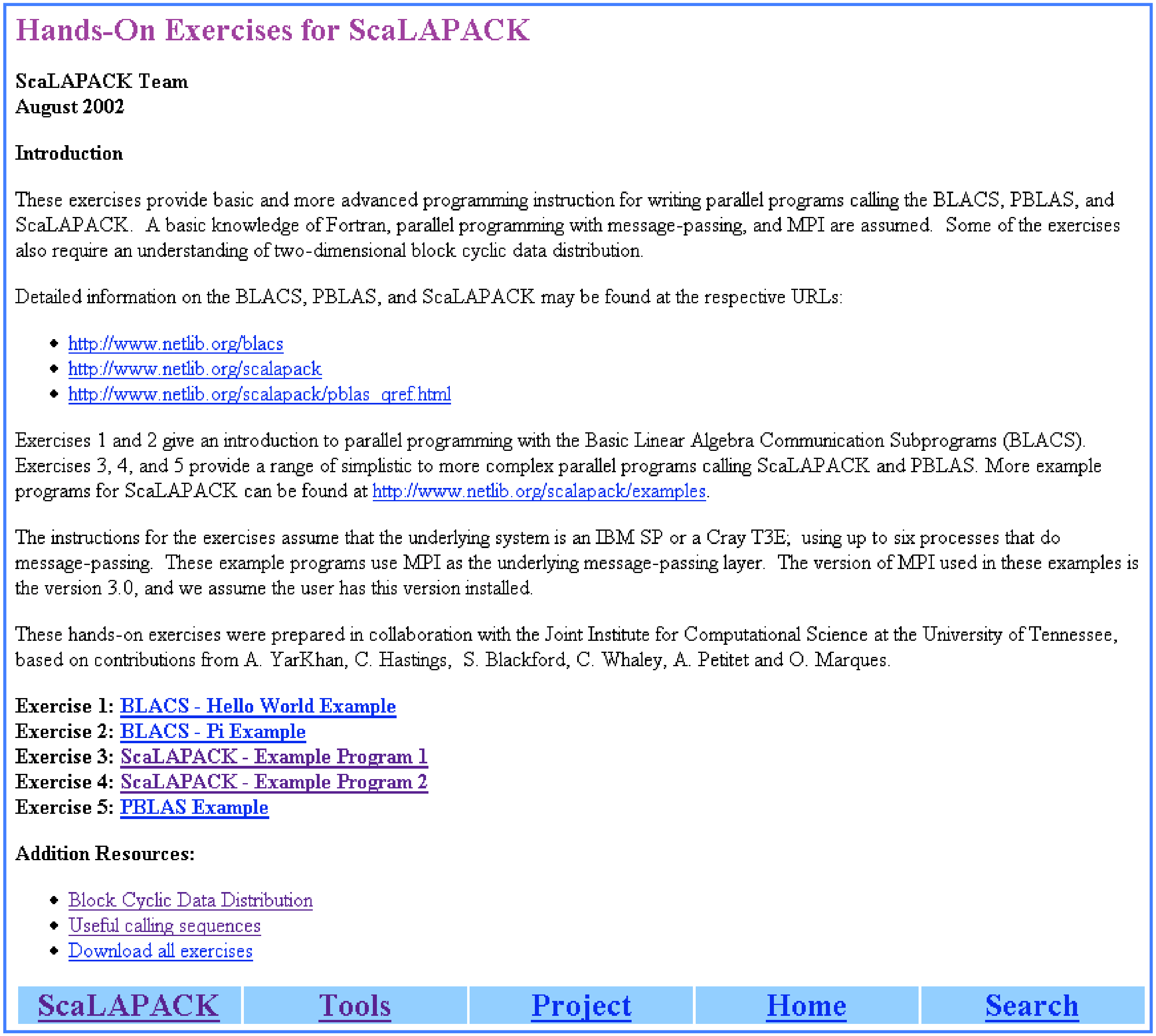
ScaLAPACK: *Performance*

- The algorithms implemented in ScaLAPACK are scalable in the sense that the parallel efficiency is an increasing function of N^2/P (problem size per node).
- Maintaining memory use per node constant allows efficiency to be maintained (in practice, a slight degradation is acceptable).
- Use efficient machine-specific BLAS (not the Fortran 77 source code available in *http://www.netlib.gov*) and BLACS (nondebug installation).
- On a distributed-memory computer:
  - Use the right number of processors
    - Rule of thumb: P=MxN/1,000,000 for an MxN matrix, which provides a local matrix of size approximately 1000-by-1000.
    - Do not try to solve a small problem on too many processors.
    - Do not exceed the physical memory.
  - Use an efficient data distribution.
    - Block size (i.e., MB,NB) = 64.
    - Square processor grid: Prow = Pcolumn.

LU/Solve on Pentium II 300 MHz Cluster



On line tutorial: *http://acts.nersc.gov/scalapack/hands-on/main.html*



Hands-On Exercises for ScaLAPACK

ScaLAPACK Team
August 2002

Introduction

These exercises provide basic and more advanced programming instruction for writing parallel programs calling the BLACS, PBLAS, and ScaLAPACK. A basic knowledge of Fortran, parallel programming with message-passing, and MPI are assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

Detailed information on the BLACS, PBLAS, and ScaLAPACK may be found at the respective URLs:

- http://www.netlib.org/blacs
- http://www.netlib.org/scalapack
- http://www.netlib.org/scalapack/pblas_qref.html

Exercises 1 and 2 give an introduction to parallel programming with the Basic Linear Algebra Communication Subprograms (BLACS). Exercises 3, 4, and 5 provide a range of simplistic to more complex parallel programs calling ScaLAPACK and PBLAS. More example programs for ScaLAPACK can be found at http://www.netlib.org/scalapack/examples.

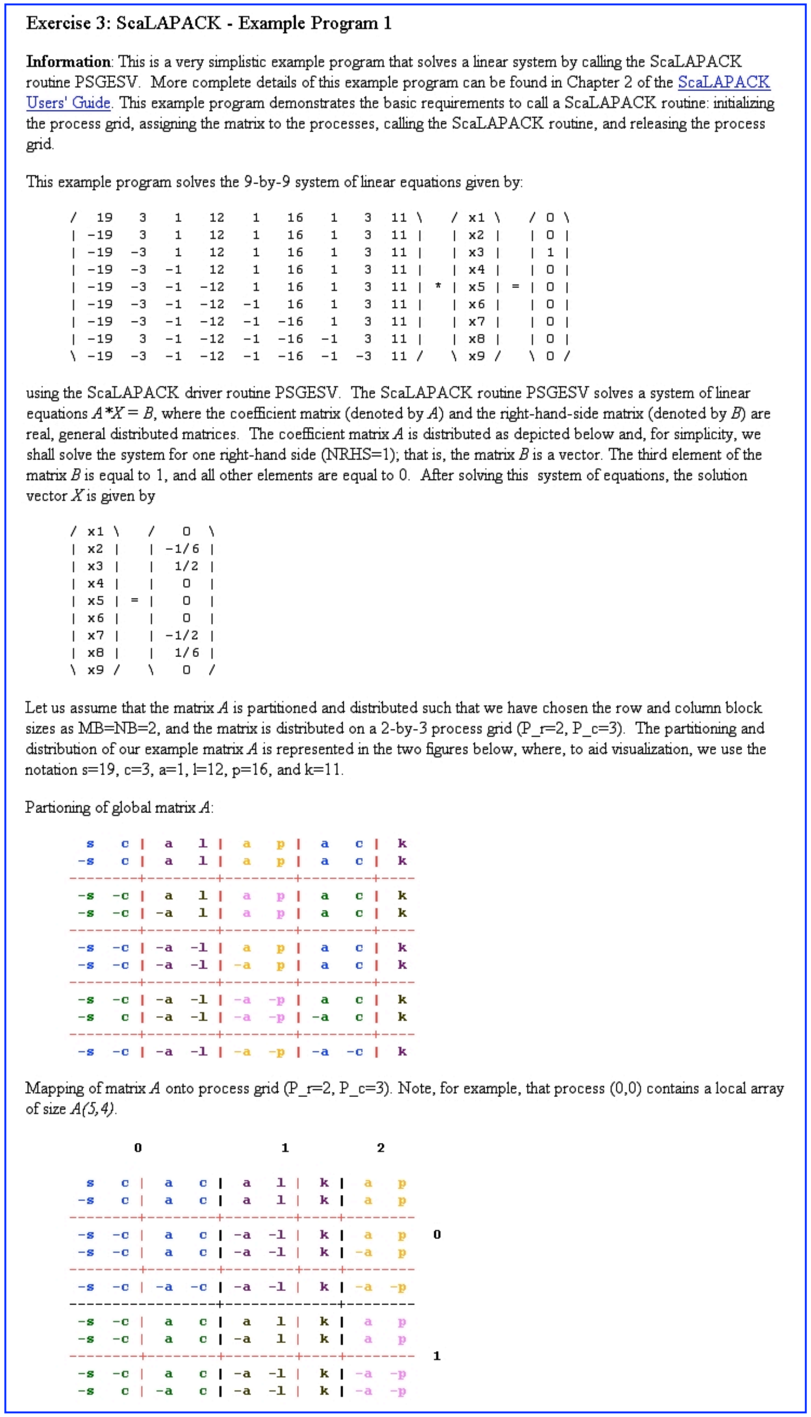
The instructions for the exercises assume that the underlying system is an IBM SP or a Cray T3E; using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. The version of MPI used in these examples is the version 3.0, and we assume the user has this version installed.

These hands-on exercises were prepared in collaboration with the Joint Institute for Computational Science at the University of Tennessee, based on contributions from A. YarKhan, C. Hastings, S. Blackford, C. Whaley, A. Petitet and O. Marques.

Exercise 1: **BLACS - Hello World Example**
Exercise 2: **BLACS - Pi Example**
Exercise 3: **ScaLAPACK - Example Program 1**
Exercise 4: **ScaLAPACK - Example Program 2**
Exercise 5: **PBLAS Example**

Addition Resources:

- Block Cyclic Data Distribution
- Useful calling sequences
- Download all exercises



Exercise 3: ScaLAPACK - Example Program 1

Information: This is a very simplistic example program that solves a linear system by calling the ScaLAPACK routine PSGESV. More complete details of this example program can be found in Chapter 2 of the ScaLAPACK Users' Guide. This example program demonstrates the basic requirements to call a ScaLAPACK routine: initializing the process grid, assigning the matrix to the processes, calling the ScaLAPACK routine, and releasing the process grid.

This example program solves the 9-by-9 system of linear equations given by:

```
/  19   3   1   12   1   16   1   3  11 \    / x1 \    / 0 \
| -19   3   1   12   1   16   1   3  11 |    | x2 |    | 0 |
| -19  -3   1   12   1   16   1   3  11 |    | x3 |    | 1 |
| -19  -3  -1   12   1   16   1   3  11 |    | x4 |    | 0 |
| -19  -3  -1  -12   1   16   1   3  11 |  * | x5 |  = | 0 |
| -19  -3  -1  -12  -1   16   1   3  11 |    | x6 |    | 0 |
| -19  -3  -1  -12  -1  -16   1   3  11 |    | x7 |    | 0 |
| -19   3  -1  -12  -1  -16  -1   3  11 |    | x8 |    | 0 |
\ -19  -3  -1  -12  -1  -16  -1  -3  11 /    \ x9 /    \ 0 /
```

using the ScaLAPACK driver routine PSGESV. The ScaLAPACK routine PSGESV solves a system of linear equations $A*X = B$, where the coefficient matrix (denoted by A) and the right-hand-side matrix (denoted by B) are real, general distributed matrices. The coefficient matrix A is distributed as depicted below and, for simplicity, we shall solve the system for one right-hand side (NRHS=1); that is, the matrix B is a vector. The third element of the matrix B is equal to 1, and all other elements are equal to 0. After solving this system of equations, the solution vector X is given by

```
/ x1 \    /   0  \
| x2 |    | -1/6 |
| x3 |    |  1/2 |
| x4 |    |   0  |
| x5 | =  |   0  |
| x6 |    |   0  |
| x7 |    | -1/2 |
| x8 |    |  1/6 |
\ x9 /    \   0  /
```

Let us assume that the matrix A is partitioned and distributed such that we have chosen the row and column block sizes as MB=NB=2, and the matrix is distributed on a 2-by-3 process grid (P_r=2, P_c=3). The partitioning and distribution of our example matrix A is represented in the two figures below, where, to aid visualization, we use the notation s=19, c=3, a=1, l=12, p=16, and k=11.

Partioning of global matrix A:

```
  s    c |  a    l |  a    p |  a    c |  k
 -s    c |  a    l |  a    p |  a    c |  k
---------+---------+---------+---------+-----
 -s   -c |  a    l |  a    p |  a    c |  k
 -s   -c | -a    l |  a    p |  a    c |  k
---------+---------+---------+---------+-----
 -s   -c | -a   -l |  a    p |  a    c |  k
 -s   -c | -a   -l | -a    p |  a    c |  k
---------+---------+---------+---------+-----
 -s   -c | -a   -l | -a   -p |  a    c |  k
 -s    c | -a   -l | -a   -p | -a    c |  k
---------+---------+---------+---------+-----
 -s   -c | -a   -l | -a   -p | -a   -c |  k
```

Mapping of matrix A onto process grid (P_r=2, P_c=3). Note, for example, that process (0,0) contains a local array of size *A(5,4)*.

```
         0                   1            2

  s    c |  a    c |  a    l |  k |  a    p
 -s    c |  a    c |  a    l |  k |  a    p
---------+---------+---------+----+---------
 -s   -c |  a    c | -a   -l |  k |  a    p     0
 -s   -c |  a    c | -a   -l |  k | -a    p
---------+---------+---------+----+---------
 -s   -c | -a   -c | -a   -l |  k | -a   -p
-------------------+--------------+---------
 -s   -c |  a    c |  a    l |  k |  a    p
 -s   -c |  a    c | -a    l |  k |  a    p
---------+---------+---------+----+---------    1
 -s   -c |  a    c | -a   -l |  k | -a   -p
 -s    c | -a    c | -a   -l |  k | -a   -p
```

```
      PROGRAM PSGESVDRIVER
*
*     Example Program solving Ax=b via ScaLAPACK routine PSGESV
*
*     .. Parameters ..

*     ..
*     .. Local Scalars ..

*     ..
*     .. Local Arrays ..

*     .. Executable Statements ..
*
*     INITIALIZE THE PROCESS GRID
*
      CALL SL_INIT( ICTXT, NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*
*     If I'm not in the process grid, go to the end of the program
*
      IF( MYROW.EQ.-1 )
     $   GO TO 10
*
*     DISTRIBUTE THE MATRIX ON THE PROCESS GRID
*     Initialize the array descriptors for the matrices A and B
*
      CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
     $               INFO )
      CALL DESCINIT( DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT,
     $               MXLLDB, INFO )
*
*     Generate matrices A and B and distribute to the process grid
*
      CALL MATINIT( A, DESCA, B, DESCB )
*
*     Make a copy of A and B for checking purposes
*
      CALL PSLACPY( 'All', N, N, A, 1, 1, DESCA, A0, 1, 1, DESCA )
      CALL PSLACPY( 'All', N, NRHS, B, 1, 1, DESCB, B0, 1, 1, DESCB )
*
*     CALL THE SCALAPACK ROUTINE
*     Solve the linear system A * X = B
*
      CALL PSGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB,
     $             INFO )
```

M

```
/**********************************************************************
/
/* This program illustrates the use of the ScaLAPACK routines PDPTTRF
*/
/* and PPPTTRS to factor and solve a symmetric positive definite
*/
/* tridiagonal system of linear equations, i.e., T*x = b, with
*/
/* different data in two distinct contexts.
*/
/**********************************************************************
/

/* a bunch of things omitted for the sake of space */

main()
{

   /* Start BLACS */
   Cblacs_pinfo( &mype, &npe );
   Cblacs_get( 0, 0, &context );
   Cblacs_gridinit( &context, "R", 1, npe );
   /* Processes 0 and 2 contain d(1:4) and e(1:4) */
   /* Processes 1 and 3 contain d(5:8) and e(5:8) */
   if      ( mype == 0 || mype == 2 ){
           d[0]=1.8180; d[1]=1.6602; d[2]=1.3420; d[3]=1.2897;
           e[0]=0.8385; e[1]=0.5681; e[2]=0.3704; e[3]=0.7027;
   }
   else if ( mype == 1 || mype == 3 ){
           d[0]=1.3412; d[1]=1.5341; d[2]=1.7271; d[3]=1.3093;
           e[0]=0.5466; e[1]=0.4449; e[2]=0.6946; e[3]=0.0000;
   }
   if ( mype == 0 || mype == 1 ) {
      /* New context for processes 0 and 1 */
      map[0]=0; map[1]=1;
      Cblacs_get( context, 10, &context_1 );
      Cblacs_gridmap( &context_1, map, 1, 1, 2 );
      /* Right-hand side is set to b = [ 1 2 3 4 5 6 7 8 ] */
      if      ( mype == 0 ) {
              b[0]=1.0; b[1]=2.0; b[2]=3.0; b[3]=4.0;
      }
      else if ( mype == 1 ) {
              b[0]=5.0; b[1]=6.0; b[2]=7.0; b[3]=8.0;
      }
      /* Array descriptor for A (D and E) */
      desca[0]=501; desca[1]=context_1; desca[2]=n; desca[3]=nb;
      desca[4]=0; desca[5]=lda; desca[6]=0;
      /* Array descriptor for B */
      descb[0]=502; descb[1]=context_1; descb[2]=n; descb[3]=nb;
      descb[4]=0; descb[5]=ldb; descb[6]=0;
      /* Factorization */
      pdpttrf( &n, d, e, &ja, desca, af, &laf,
               work, &lwork, &info );
      /* Solution */
      pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
```

```
else {
      /* New context for processes 0 and 1 */
      map[0]=2; map[1]=3;
      Cblacs_get( context, 10, &context_2 );
      Cblacs_gridmap( &context_2, map, 1, 1, 2 );
      /* Right-hand side is set to b = [ 8 7 6 5 4 3 2 1 ] */
      if      ( mype == 2 ) {
              b[0]=8.0; b[1]=7.0; b[2]=6.0; b[3]=5.0;
      }
      else if ( mype == 3 ) {
              b[0]=4.0; b[1]=3.0; b[2]=2.0; b[3]=1.0;
      }
      /* Array descriptor for A (D and E) */
      desca[0]=501; desca[1]=context_2; desca[2]=n; desca[3]=nb;
      desca[4]=0; desca[5]=lda; desca[6]=0;
      /* Array descriptor for B */
      descb[0]=502; descb[1]=context_2; descb[2]=n; descb[3]=nb;
      descb[4]=0; descb[5]=ldb; descb[6]=0;
      /* Factorization */
      pdpttrf( &n, d, e, &ja, desca, af, &laf,
               work, &lwork, &info );
      /* Solution */
      pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
               af, &laf, work, &lwork, &info );
      printf( "MYPE=%i: x[:] = %7.4f %7.4f %7.4f %7.4f\n",
              mype, b[0], b[1], b[2], b[3]);
   }
   Cblacs_gridexit( context );
   Cblacs_exit( 0 );
}
```

```
Using Matlab notation:

T = diag(D)+diag(E,-1)+diag(E,1)

where

D = [ 1.8180 1.6602 1.3420 1.2897 1.3412 1.5341 1.7271 1.3093
]
E = [ 0.8385 0.5681 0.3704 0.7027 0.5466 0.4449 0.6946 ]

Then, solving T*x = b,

if b = [ 1 2 3 4 5 6 7 8 ]
x = [ 0.3002 0.5417 1.4942 1.8546 1.5008 3.0806 1.0197 5.5692
]

if b = [ 8 7 6 5 4 3 2 1 ]
x = [ 3.9036 1.0772 3.4122 2.1837 1.3090 1.2988 0.6563 0.4156
]
```

ScaLAPACK: *Commercial Use*

ScaLAPACK has been incorporated in the following commercial packages:

- Fujitsu
- Hewlett-Packard/Convex
- Hitachi
- IBM Parallel ESSL
- NAG Numerical PVM (and MPI) Library
- Cray LIBSCI
- NEC Scientific Software Library
- Sun Scientific Software Library
- Visual Numerics (IMSL)

ScaLAPACK: *Development Team*

- Susan Blackford, UTK
- Jaeyoung Choi, Soongsil University
- Andy Cleary, LLNL
- Ed D'Azevedo, ORNL
- Jim Demmel, UCB
- Inderjit Dhillon, UT Austin
- Jack Dongarra, UTK
- Ray Fellers, LLNL
- Sven Hammarling, NAG
- Greg Henry, Intel
- Osni Marques, LBNL/NERSC
- Caroline Papadopoulos, UCSD
- Antoine Petitet, UTK
- Ken Stanley, UCB
- Francoise Tisseur, Manchester
- David Walker, Cardiff
- Clint Whaley, UTK